Learning to Generalize

AW

Lecture Overview

1. **Bias-Variance Tradeoff**

2. **Tuning and Evaluation**

3. **Parameter Norm Penalties**

4. **Ensemble Methods + Early Stopping**

Generalization (again)

- In a machine learning problem, we try to generalize the known dependent variable on seen instances to unseen instances.
  - Unseen $\Rightarrow$ The model did not see it during training.
- Generalization= given training images with seen labels, try to label an unseen image.
- The classification accuracy on instances used to train a model is usually higher than on unseen instances.
  - That is if we do not take special efforts to regularize
- We only care about the accuracy on unseen data.

Generalization – Reasons for Overfitting

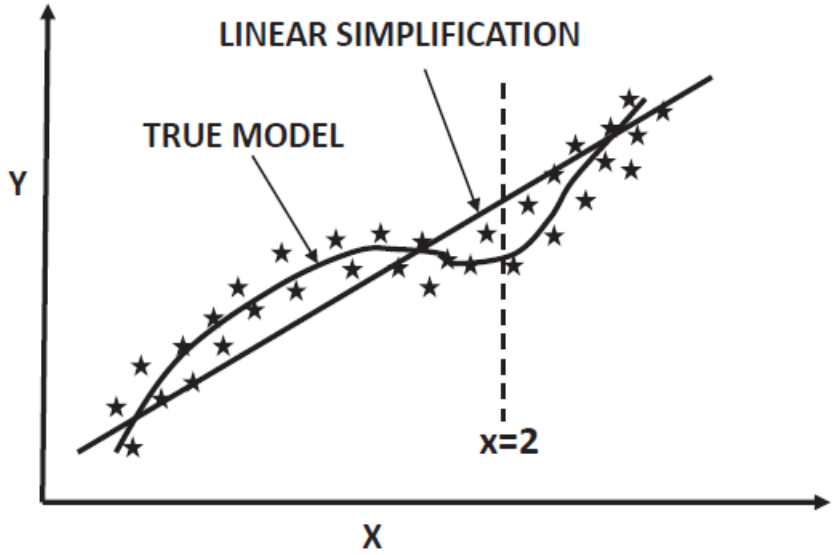
- Why is the accuracy on seen data higher?
  - Trained model remembers some of the irrelevant nuances.
  - Regularization goal is to defeat it
- When is the gap between seen and unseen accuracy likely to be high?

Generalization – Reasons for Overfitting

- Why is the accuracy on seen data higher?
  - Trained model remembers some of the irrelevant nuances.
  - Regularization goal is to defeat it
- When is the gap between seen and unseen accuracy likely to be high?
  - When the amount of data is limited.
  - When the model is complex (which has higher *capacity* to remember nuances).
  - The combination of the two is a deadly cocktail.
- A high accuracy gap between the predictions on seen and unseen data is referred to as *overfitting*.

Data and Models

Example: True data and 2 models- polynomial and linear



First impression: Polynomial model such as

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

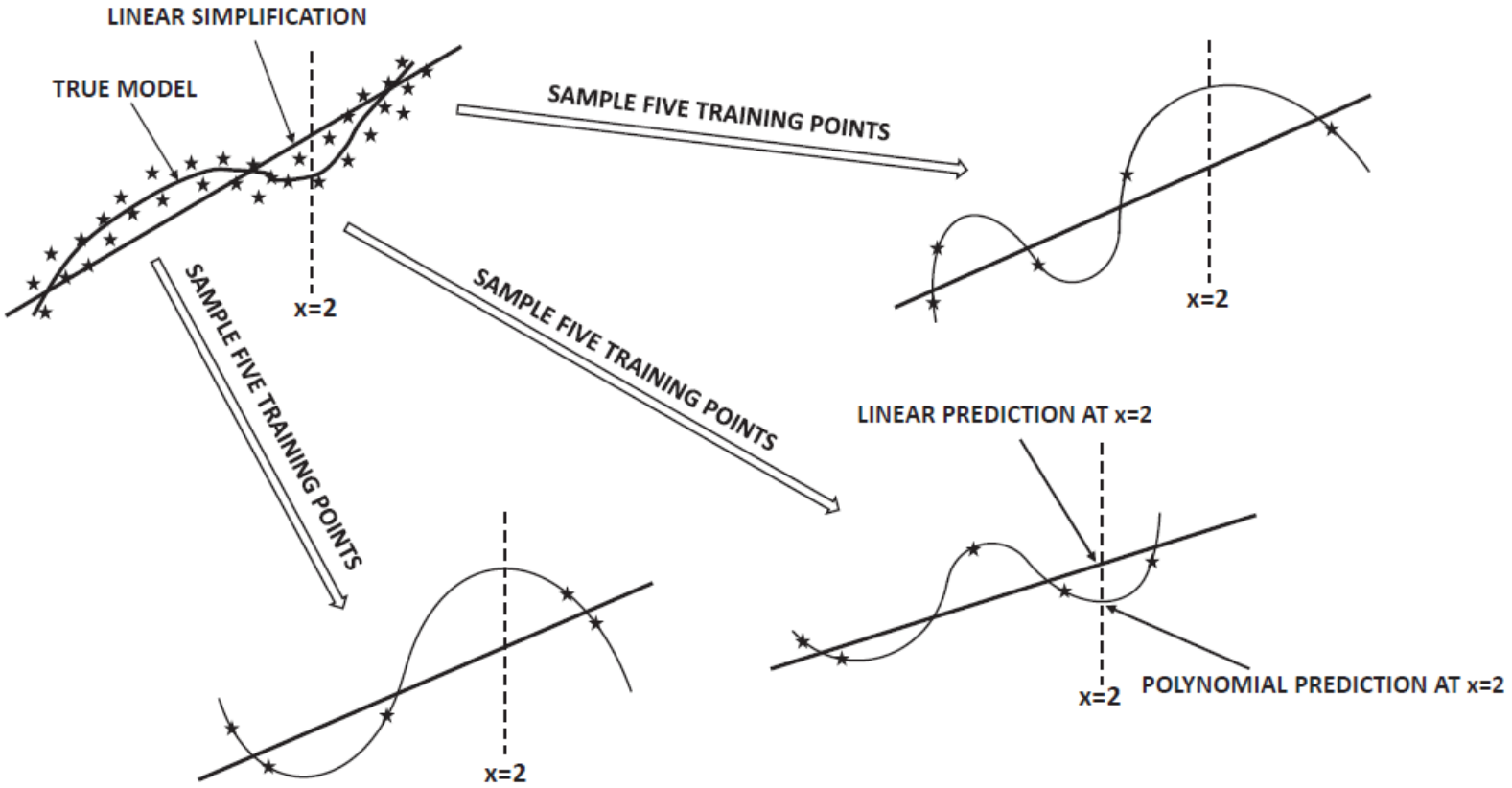
is "better" than linear model

$$y = w_0 + w_1 x$$

Bias-variance trade-off says: "Not necessarily! Are you looking at all data or just some sample?"

Data and Models (cont.)

Example: Polynomial model vs linear



Second look: Zero error on training data, but wildly varying predictions of $x = 2$

Bias and Variance -Intuition

- The *bias* error is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)
  - It is an error caused by rigid assumptions upfront into a model/low flexibility that do not allow to match training data
  - Bias is manifesting in the big difference between the average prediction of our model and the correct value in training data which we are trying to predict.

Problem: little attention to training data

- Model with high bias has either not enough training parameters to match data or has domain of the parameters set so that they cannot fit training data
- It is often (but not always) associated with oversimplifying

Bias and Variance -Intuition

The *variance* is an error from sensitivity to small fluctuations in the training set: the model is too flexible and adapts to all minor changes in training data set.

- High variance may result from an algorithm with many parameters modeling the random noise in the training data rather than capture data model

Example:

- The higher-order polynomial model is more flexible than the linear model (more weights to train), so it has less bias.
- It has more parameters, so
  - For a small training data set, the learned parameters will be more sensitive to the nuances of that data set.
  - Different training data sets will provide different values of weights and thus, predictions for y at a particular x will depend too much on training data set.
  - This variation between training data sets guarantees high variance

Bias and Variance -Intuition

- Intuitively variance how much the learning method will move around its mean prediction over all possible data sets
- Intuitively bias is how much the model misses the data for all possible data sets.
- The bias–variance tradeoff is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data.
- Neural networks are inherently low-bias and high-variance learners $\Rightarrow$ Need ways of handling complexity.

Assumptions About Data

- The relationship between the dependent variable y and its feature representation $\vec{x}$ is given by some unknown function and additive noise ϵ
- The true model is $y = f(\vec{x})$ where $y \in Y, x \in X$ are range and domain of the function which needs to be learned. Noise refers to unexplained variations of data from true model so that data is given by $y_i = f(\vec{x}_i) + \epsilon_i$ for data points $(x_i, y_i) \in D \subseteq X \times Y,\ i \in \{1, \dots, N\}$. Noise is a property of the *data* rather than model.
- Noise is a random variable with 0 expectation, so the model $y = f(\vec{x})$ together with noise defined true distribution B on $X \times Y$
- Real-world noise examples:
  - Human mislabeling of test instance $\Rightarrow$ Ideal model will never predict it accurately.
  - Error during collection of temperature due to sensor malfunctioning.
- Cannot do anything about it even if seeded with knowledge about true model.

Bias-Variance Imaginable Experiment

- Imagine you are given the true distribution B of training data (including labels).
- You have a principled way of sampling data sets $\mathcal{D} \sim B$ from the training distribution.
- Imagine you create an infinite number of training data sets $\mathcal{D}_i$ (and trained models $A(\mathcal{D}_i)$) by repeated sampling.
- You have a *fixed* set $\mathcal{T}$ of unlabeled test instances.
  - The test set $\mathcal{T}$ does not change over different training data sets.
- Compute prediction $g(x_j, A(\mathcal{D}_i))$ of each instance x_j in $\mathcal{T}$ for each trained model $A(\mathcal{D}_i))$.

Informal Definition of Bias

- Compute averaged prediction of each test instance x over different training models $G_A(x)$(something like $\lim_{n\to\infty} \frac{1}{n}\sum_i g(x, A(\mathcal{D}_i))$)
- Averaged prediction of test instance will be different from true (unknown) model $f(x)$. Difference between (averaged) $G_A(x)$ and $f(x)$ caused by erroneous assumptions/simplifications in modeling $\Rightarrow$ Bias
- High bias = underfitting!

Example (cont.): Linear simplification to polynomial model causes bias.

- If the true (unknown) model $f(x)$ were an order-4 polynomial, and we used any polynomial of order-4 or greater in $G_A(x)$ bias would be 0.

Informal Definition of Variance

- The value $g(x, A(\mathcal{D}_i))$ will vary with $\mathcal{D}_i$ for fixed x.
  - The prediction of the same test instance will be different over different trained models.
- Variance of $g(x, A(\mathcal{D}_i))$ over different training data sets $\mathcal{D}_i \Rightarrow$ *Model Variance*
- All these predictions cannot be simultaneously correct $\Rightarrow$ *v*ariation contributes to error
- High variance = overfitting!

Example (cont.): Linear model will have low model variance.

- Higher-order model will have high variance.

Formal Bias-Variance Equation for MSE

- Theorem. The expectation over different choices of the training set $D = \{(x_1, y_t), \ldots (x_t, y_t)\}$ all sampled from the same joint distribution $D \sim B$ is computed as

$$E_{D\sim B,\epsilon}\left((g(\vec{x}, A\,(D)) - y)^2\right) = \underbrace{\frac{1}{t}\sum_i \left(f(\vec{x}_i) - E\left(g\left(\vec{x}_i, A(\mathcal{D})\right)\right)\right)^2}_{bias^2}$$

$$+ \underbrace{\frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - E\left(g\left(\vec{x}_i, A(\mathcal{D})\right)\right)\right)^2\right)}_{variance}$$

$$+ \underbrace{\frac{\sum_i E(\epsilon_i^2)}{\mathrm{t}}}_{noise}$$

Formal Bias-Variance Equation for MSE

$$MSE(D, A(\mathcal{D})) = \frac{1}{t}\sum_i (\hat{y}_i - y_i)^2 = \frac{1}{t}\sum_i \left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i) - \epsilon_i\right)^2 \text{ so}$$

$$E_{\mathcal{D}\sim B}(MSE(D, A(\mathcal{D})) = E\left(\frac{1}{t}\sum_i \left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i) - \epsilon_i\right)^2\right)$$

$$= \frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i) - \epsilon_i\right)^2\right)$$

$$= \frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right)^2 - 2\epsilon_i\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right) + \epsilon_i^2\right)$$

$$= \frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right)^2\right) - E\left(2\epsilon_i\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right)\right) + E(\epsilon_i^2)$$

Since ϵ_i is independent of $g\left(\vec{x}_i, A(\mathcal{D})\right)$ which is determined by B we have $E\left(2\epsilon_i\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right)\right) = 2E(\epsilon_i)E\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i)\right) = 0$ as $E(\epsilon_i) = 0$ which is our assumption, so

$$E_{D\sim B}\left(MSE\left(D, A(\mathcal{D})\right)\right) = \frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f\left(\vec{x}_i\right)\right)^2\right) + \frac{1}{\mathrm{t}}\sum_i E(\epsilon_i^2)$$

Formal Bias-Variance Equation for MSE (cont.)

$$E_{D\sim B}\left(MSE\left(D,A(\mathcal{D})\right)\right)=E\left(\frac{1}{t}\sum_i(\hat{y}_i-y_i)^2\right)=E\left(\frac{1}{t}\sum_i\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)-\epsilon_i\right)^2\right)$$

$$=\frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)\right)^2\right)+\frac{1}{\mathrm{t}}\sum_i E(\epsilon_i^2)$$

$$=\frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)+E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)-f(\vec{x}_i)\right)^2\right)+\frac{\sum_i E(\epsilon_i^2)}{\mathrm{t}}$$

$$=\frac{1}{t}\sum_i E\left(\left(f(\vec{x}_i)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)\right)^2\right)+\frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)\right)^2\right)$$

$$-\frac{2}{t}\sum_i E\left[\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)\right)\cdot\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)\right]+\frac{\sum_i E(\epsilon_i^2)}{\mathrm{t}}$$

Clearly terms of the product in the third term (prediction error and prediction difference from its expectation) are independent so

$$E\left[\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)\right)\cdot\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)\right]=$$

$$E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)\right)\cdot E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)\right)\right)$$

$$=E\left(g\left(\vec{x}_i,A(\mathcal{D})\right)-f(\vec{x}_i)\right)\cdot 0=0$$

Formal Bias-Variance Equation for MSE (cont.)

So $E\left(MSE\left(D, A(\mathcal{D})\right)\right) = \frac{1}{t}\sum_i (\hat{y}_i - y_i)^2 = \frac{1}{t}\sum_i \left(g\left(\vec{x}_i, A(\mathcal{D})\right) - f(\vec{x}_i) - \epsilon_i\right)^2$

$$= \underbrace{\frac{1}{t}\sum_i \left(f(\vec{x}_i) - E\left(g\left(\vec{x}_i, A(\mathcal{D})\right)\right)\right)^2}_{bias^2}$$

$$+ \underbrace{\frac{1}{t}\sum_i E\left(\left(g\left(\vec{x}_i, A(\mathcal{D})\right) - E\left(g\left(\vec{x}_i, A(\mathcal{D})\right)\right)\right)^2\right)}_{variance}$$

$$+ \underbrace{\frac{\sum_i E(\epsilon_i^2)}{\mathrm{t}}}_{noise}$$

Bias-Variance Equation Interpretation

- $E[MSE]$ is the expected mean-squared error of the fixed set of test instances over different samples of training data sets.

  $E[MSE]$ = Bias^2 + Variance +Noise

  - In linear models, the bias component will contribute more to $E[MSE]$.
  - In polynomial models, the variance component will contribute more to $E[MSE]$.
- We have a trade-off, when it comes to choosing model complexity!

Main Lessons form Bias-Variance Analysis

- A model with greater complexity might be *theoretically* more accurate (i.e., low bias).
  - But you have less control on what it might predict on a small training data set.
  - Different training data sets will result in widely *varying* predictions of same test instance.
  - Some of these must be wrong ⇒ Contribution of model variance.
- *A more accurate model for infinite data is not a more accurate model for finite data.*
  - Do not use a sledgehammer to swat a fly!

Lecture Overview

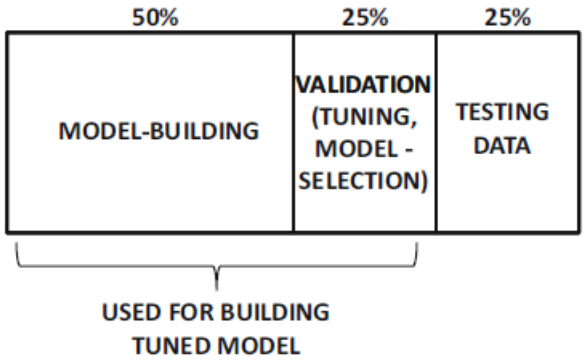
1. Bias-Variance Tradeoff Formally
2. Tuning and Evaluation
3. Parameter Norm Penalties
4. Ensemble Methods + Early Stopping

Data: Training, Validation and Testing

Training data: This part of the data is used to build the training model

- Completely different NN algorithms may be used to build the models in multiple ways.
  - e.g. IMDB sentiment analysis can be trained with recurrent, convolution and FF networks
  - The same convolutional network may use different number of neurons in each layer for image recognition, etc.
- The same neural network might use different hyperparameters for the learning rate or for regularization to create different models.
- The same training data set may be tried multiple times over different choices for the hyperparameters or

Data: Training, Validation and Testing (cont.)



Validation data: Data set used model for selection and parameter tuning.

- *Model selection*, is the process of selection of best algorithm among different models. Actual *evaluation* of these algorithms for selecting the best model is done on a separate validation data set to avoid favoring overfitted models
  - For example, the choice of the learning rate may be tuned by constructing the model multiple times on the training data, and using the same validation set to estimate the accuracy of these different models.
- Validation data is the part of model building

Data: Training, Validation and Testing (cont.)

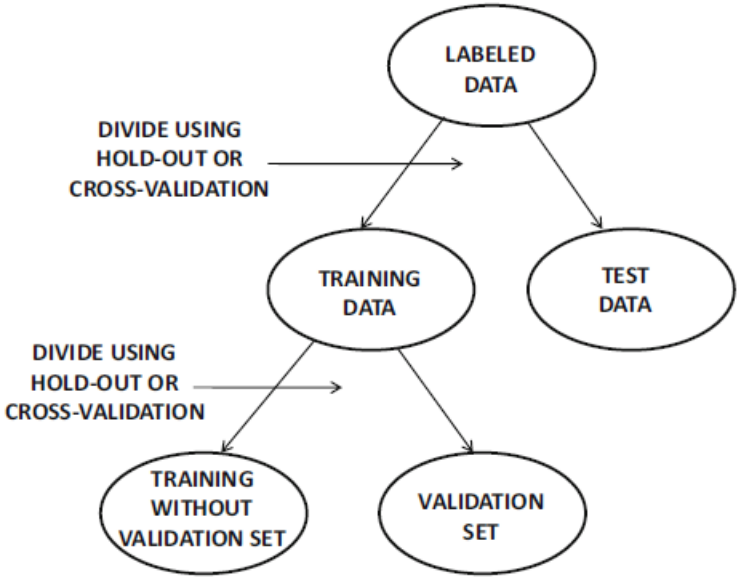
- *Testing data:* It is used to test the accuracy of the final (tuned) model. Testing data are not used during the process of parameter tuning and model selection to prevent overfitting. These data are *used only once at the very end of the process.* It is an extraordinarily strict and important requirement
  - If the results on the test data are used to adjust the model, then the results of adjustments are contaminated with knowledge from the testing data.
- When only small data sets are available the rule of thumb for data division is:

  training data: validation data: testing data = 2:1:1
- When large data sets are available (big data case) it is not uncommon to use 98:1:1 division of data

Holdouts and Cross-Validation

Hold-out method:

- a fraction of the instances are used to build the training model.
- The remaining instances (*held-out* Instances) are used for testing. The accuracy of predicting the labels of held-out instances is reported as the overall accuracy.



Cross-validation method:

- Labeled data is divided into q equal segments. One of q sets is used for testing, the remaining $(q-1)$ segements are used for training.
- This process is repeated q times by using each of the q segments as the test set. The average accuracy over the q different test sets is reported.

Holdouts and Cross-Validation: Pros and Cons

Hold-out method:

- underestimates the true accuracy: frequency of the held-in examples will always be inversely related to that of the held-out examples ⇒ leads to a pessimistic bias in the evaluation.
  - Unless we sample each class separately wrt probabilities, the held-out examples have a higher presence of a particular class than all labeled data set ⇒ held-in examples have a lower average presence of the same class. Causes a mismatch between the training and test data
  - Simple and efficient

Cross-validation:

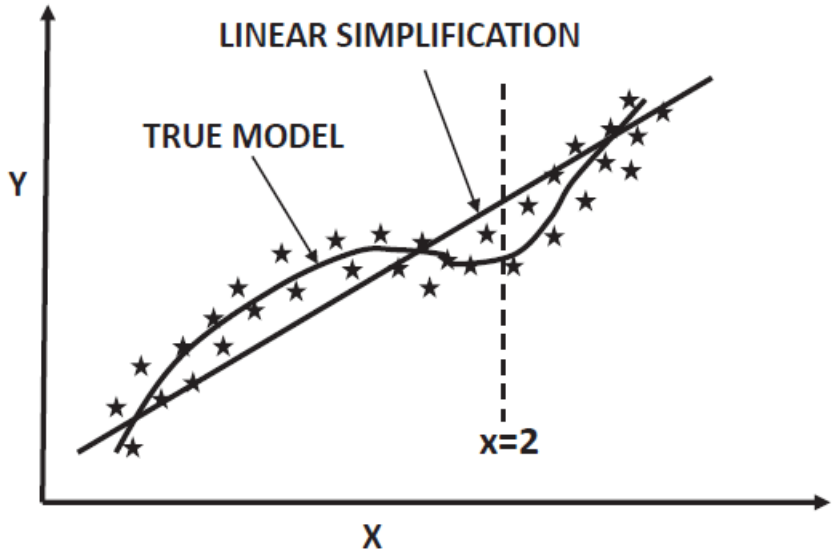
- Can closely approximate the accuracy,
- it is usually too expensive to train the model a large number of times. Cross-validation is sparingly used in neural networks because of efficiency issues.

Lecture Overview

1. Bias-Variance Tradeoff Formally
2. Tuning and Evaluation
3. Parameter Norm Penalties
4. Ensemble Methods + Early Stopping

Data and Models Again

Example:



First impression: Polynomial model such as

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

is "better" than linear model $y = w_0 + w_1 x$.

We have shown not – very often the case!

Why? *High variance!*

Economy in Parameters

High variance possible because we have many parameters to tune!

- A lower-order model has economy in parameters.
  - A linear model uses two parameters, whereas an order-4 model uses five parameters.
- Economy in parameters discourages overfitting.
- Choosing a neural network with fewer units per layer enforces economy.
- But fixing the architecture up front is an inflexible solution.
  - What if these parameters bring much better model?

Soft Economy vs Hard Economy

- A softer solution uses a larger model but imposes a (tunable) penalty on parameter use. For example:

$$\hat{y} = \sum_{i=0}^{d} w_i x^i$$

- Loss function:

$$L = \sum_{(x,y)\in D} (y - \hat{y})^2 + \underbrace{\lambda \sum_{i=0}^{d} w_i^2}_{L^2-regularization}$$

- The (tuned) value of λ decides the level of regularization.
- Softer constraint approach with a complex model performs better than rigid architecture!

Effect of Soft Constraints on Updates

- For learning rate α, effect on update is to multiply parameter with $(1 - \alpha\lambda) \in (0, 1)$

$$w_i \Leftarrow w_i(1 - \alpha\lambda) - \alpha \frac{\partial L}{\partial w_i}$$

- Interpretation: Decay-based forgetting!
- Unless a parameter is important, it will have small absolute value.
  - Model decides what is important.
  - Better than inflexibly deciding up front.

L^1-regularization

- Model gives on parameter use.

$$\hat{y} = \sum_{i=0}^{d} w_i x^i$$

- Loss function:

$$L = \sum_{(x,y)\in D} (y - \hat{y})^2 + \underbrace{\lambda \sum_{i=0}^{d} |w_i|}_{L^1-regularization}$$

- Update has slightly different form for learning rate $\alpha \in (0,1)$:

$$w_i \Leftarrow w_i - \alpha\lambda s_i - \alpha \frac{\partial L}{\partial w_i}$$

where s_i is the sign of w_i that comes from derivative of $|w_i|$ w.r.t w_i so

$$s_i = \begin{cases} -1 \text{ when } w_i < 0 \\ 1 \text{ when } w_i \geq 0 \end{cases}$$

- L^1-regularization leads to sparse parameter learning.
  - Zero values of w_i can be dropped.
  - Equivalent to dropping edges from neural network.
- L^2-regularization generally provides better performance.

Penalizing Hidden Neurons

- So far penalize the *parameters* of the neural network.
- Can penalize the *activations* of the neural network, so that only a small subset of the neurons are activated for any given data instance, meaning that
  - The neural network may be large and complex, but only a small part of it is active in predicting on given data instance
- To achieve sparsity can impose an L^1-penalty on the hidden units:

$$L' = L + \lambda \sum_{i=1}^{M} |y_{h_i}|$$

where L is original loss function and $|y_{h_i}|$ is the magnitude of output of h_i

- The backward gradient flow is then recomputed respectively

Lecture Overview

1. Bias-Variance Tradeoff Formally

2. Tuning and Evaluation

3. Parameter Norm Penalties

4. Ensemble Methods + Early Stopping

Ensemble Methods - Reminder

Idea: If enough training data is available then take different subsets of training data and train model on each one. Then make a prediction on test instance using each model separately. Then 'average' the predictions to create the final prediction.

- One can truly average the real-valued predictions (probability of outcome with softmax) or take the median (when regression).
  - If probabilistic predictions are averaged, it is common to average the *logarithms* of these values which correspond to taking geometric mean of original values
- In case of the discrete predictions majority voting can be used instead of true averaging

Bagging in Detail

- The training data is sampled with replacement.
  - The sample size s may be different from the size of the training data size n. Often best results are obtained when $s \sim \frac{3}{4} n$
  - Yet, it is common to set s to n. If so the resampled data will contain duplicates, and about a fraction $\left(1 - \frac{1}{n}\right)^n \approx 1/e$ of the original data set will not be included at all.
- A model is constructed on the resampled training data set
- The entire process of resampling and model building is repeated m times.
- For a given test instance, each of these m models is applied to the test data.
- The predictions from different models are then averaged

Subsampling and Bucket-of-Models

- With subsampling different models are constructed on the samples of the data created *without* replacement.
- The predictions from the different models are averaged.
  - Clearly here sample size $s < n$, because if $s = n$ then the training data is always the same, so we'll get identical results across different ensemble components.

Bucket-of-models can be used with subsampling or bagging.

- The idea is to have different models, not only different parameter values. Most often this is done by having different meta parameters (such as learning rate and regularization weights).
- k best configurations and then average the predictions of these configurations

Early Stopping

Executing gradient descent to convergence optimizes the loss on the training data, but not necessarily on the out-of-sample test data:

- final few steps often overfit to the specific nuances of the training data, which might not generalize well to the test data

Early Stopping:

- Portion of the training data is held out as a validation set. The backpropagation-based training is only applied to the portion of the training data that does not include the validation set.
- The error of the model on the validation set is continuously monitored. If there is a point at which the error on begins to rise monotonously on the validation set, even though it continues to reduce on the training set, then from this point on further training causes overfitting. Therefore, this point can be chosen for termination.

Reading

Sections 4.1, 4.2, 4.3.1, 4.4, 4.5.1, 4.6